Changes over Levon's extension

May 22, 2014

New extension is compared to Levon's work ([2]), mostly chapter 4

Domains Geometry Definition

Originally

see [2] – 4.3.1.1 and 4.3.1.2

Saldamli defines domain shape by listing its boundaries. Individual boundaries (points in 1D, curves in 2D resp. surfaces in 3D) are describes by shape-functions. Shape-function maps intervals ([0,1] in 2D – boundary is a curve, [0,1]×[0,1] in 3D – a surface) onto the boundary.

Example circular domain

```
class Circle
  extends Boundary(ndims=2);
  parameter Point c = {0,0};
  parameter Real r = 1;
  redeclare function shape
    input Real tau;          //tau in [0,1]
    output Real coord[2];
  algorithm
    coord := c + r * { cos(2*Pi*tau), sin(2*Pi*tau) };
  end shape;
end Circle;


type CircularDomain
  extends Cartesian2D(boundary = {circle});
  parameter Point center;
  parameter Real radius;
  parameter Circle circle (c = center, r = radius);
end CircularDomain;
```

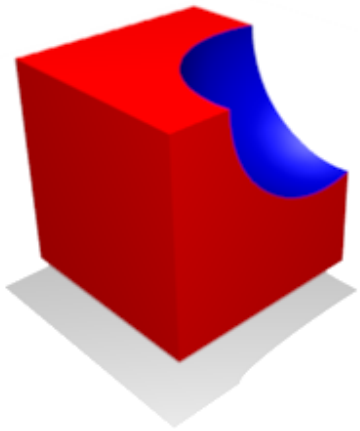

Figure 1: Boundary in 3D

Problem

This approach doesn't work well in 3D: if boundary is made of several surfaces, parameters (arguments) of shape-functions of these surfaces must be bounded not just in [0,1] intervals but in some more complex sets to compose continuous boundary, e.g. see fig. 1. And there is no way to write this in Levon's extension.

There is also no simple way to generate grid points during translation/solution.

Alternative approach

According to Peter's book [1] – 8.5.2, define interior and boundaries of domain (these elements we call *regions* here) with one shape-function and for each region specify intervals for the shape-function arguments. This approach isn't more general, but is consistent in 1, 2 and 3D and (to me) is more natural. Inner points may be generated using this shape-function.

Modified `Domain` built-in type:

```
type Domain
  parameter Integer ndims;
  Real cartesian[ndims];
  Real coord[ndims] = cartesian;
  replaceable Region interior;
  replaceable function shape
    input Real u[ndims];
    output Real coord[ndims];
  end shape;
end Domain;
```

`Region` built-in type instead of `Boundary`:

```
type Region
  parameter Integer ndim;
  replaceable function shape;
    input Real u[ndim];
    output Real coord[ndim];
  end shape;
  parameter Real[ndim][2] interval;
end Region;
```

Example circular domain:

```
class DomainCircular2D
  extends Domain;
  parameter Real radius;
  parameter Real[2] c = {0,0};
  function shapeFunc
    input Real r,v;
    output Real coord[2];
  algorithm
    coord := c + radius * r * { cos(2*Pi*v), sin(2*Pi*v) };
  end shapeFunc;
  Region2D interior(shape = shapeFunc, interval = {{0,1},{0,1}});
  Region1D boundary(shape = shapeFunc, interval = {1,{0,1}});
end DomainCircular2D;
```

Modified version

use equations instead of shape-function

```
class DomainCircular2D
  extends Domain;
  parameter Real radius = 1;
  parameter Real c = {0,0};
  Real r, theta;
  Region2D interior(theta in (0,2*C.pi), r in (0,radius));
  Region1D boundary(theta in (0,2*C.pi), r = radius);
equation
  //coordinate transformation equation:
  coord = c + r * { cos(theta), sin(theta) };
end DomainCircular2D;
```

More complex geometries

More complex geometries may be defined using *Constructive Solid Geometry* – it is applying union, intersection and difference on previously defined shapes. The syntax is not designed already. It should be also possible to define domain in external file from some CAD app.

References

[1] Peter Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1.* Wiley-IEEE Press, 2004.

[2] Levon Saldamli. *A High-Level Language for Modeling with Partial Differential Equations.* PhD thesis, Department of Computer and Information Science, Linköping University, 2006.